386 | ASM
Release Notes for Version 4.1

Phar Lap Software, Inc.
60 Aberdeen Ave., Cambridge, MA 02138
(617) 661-1510
FAX (617) 876-2972
tech-support@pharlap.com

First Edition: January 1992

386 | ASM Release Notes for Version 4.1

This file documents differences between the 4.1 version of 386 | ASM and the second edition of the *386 | ASM Reference Manual,* published in January 1991.

Summary of New Features

Command Line Switches

-[80]386R	Specify target CPU of 80386 real mode.
-[80]486	Specify target CPU of 80486.
-[80]486P	Specify target CPU of 80486 with privileged instructions.
-[80]486R	Specify target CPU of 80486 real mode.
-CV[SYM]	Generate CodeView symbol information in the object file. CodeView symbol records are used for source code debugging.
-LN[UM]	Generate line number records in the object file. Line number records are used for source code debugging.
-NOCV[SYM]	Do not generate CodeView symbol information in the object file. This is the default.
-NOLN[UM]	Do not generate line number records in the object file. This is the default.

-NOSIGNLOGI	Disable generation of undocumented form of arithmetic instructions.
-PUBPARSE c	Remove specified character from front of public symbol definition records.
-SIGNLOGI	Enable generation of undocumented form of arithmetic instructions. This is the default.

Directives

.386R	Specify target as 386 real mode.
.486	Specify target as 486.
.486P	Specify target as 486 with privileged instructions.
.486R	Specify target as 486 real mode.
COMM {NEAR|FAR} name size{:count}	Allocate communal variables at link time.
.NOSIGNLOGI	Disable generation of undocumented form of arithmetic instructions.
.PUBPARSE c	Remove specified character from front of public symbol definition records.
.SIGNLOGI	Enable generation of undocumented form of arithmetic instructions.

Applications Instructions

BSWAP	Byte swap between big and little endian.
XADD	Exchange and add.
CMPXCHG	Compare and exchange.

System Instructions

INVD	Invalidate data cache.
WBINVD	Write back and invalidate data cache.
INVLPG	Invalid TLB entry.

Test Registers

TR3, TR4 and TR5

Support for Source Code Debugging

386 | ASM can now generate source debug information for use by source code debuggers. This information is automatically included by 386 | LINK in output symbol table formats that can represent source debug information (see the *386 | LINK Reference Manual*). Generation of source debug information is controlled with two pairs of switches:

-LNUM	Causes line number records to be included in the object file.
-NOLNUM	Disables generation of line number records; this is the default.
-CVSYM	Causes CodeView symbol information to be included in the object file, and also causes line number records to be included in the object file even if -NOLVNUM is selected.
-NOCVSYM	Disables generation of CodeView symbol information; this is the default.

Support for i486 Instruction Set

Support has been added for the new i486 instructions as described in the *i486 Microprocessor Handbook*. The new applications instructions are:

BSWAP	Byte swap between big and little endian.
XADD	Exchange and add.
CMPXCHG	Compare and exchange. The base opcode for the 80486 CMPXCHG instruction was changed from 0FA6h to 0FB0h from step A to step B of the 486. 386 | ASM produces the 0FB0 form of the opcode which is valid on all 80486 CPUs which are step B1 or later.

The new system instructions are:

INVD	Invalidate data cache.
WBINVD	Write back and invalidate data cache.
INVLPG	Invalid TLB entry.

In addition, three new test registers are available when assembling privileged instructions for an 80486 target: TR3, TR4, and TR5. These new instructions and registers can be enabled with both command line switches and assembler directives. The new command line switches are:

- -80486
- -80486P
- -80486R

The new assembler directives are:

- .486
- .486P
- .486R

The new switches and directives are used in exactly the same way as their 80386 counterparts (-80386, .386, etc.) as documented in the *386 | ASM Reference Manual.*

Support for Real-Mode 80386 Programs to Use 32-Bit Instruction Operands and Addressing Modes

A new directive and a command line switch have been added to 386 | ASM to support programs which will execute in real mode and want to use the 32-bit registers and addressing modes of the 386. They enable the full 80386 instruction set, but cause the assembler to force all segments for the assembly to be USE16. The assembler generates all the necessary override bytes for the 32-bit instructions, and creates an 8086 style object module. This object module can then be linked into an existing real mode program with any existing 8086 linkers. Thus a module which contains 32-bit arithmetic routines could be added to an existing 8086 program.

The command line switch which enables 80386 instructions for real mode is called -386R. It is specified in the same way as any of the other CPU switches. The equivalent assembler directive is named .386R, and is subject to the same positioning rules as the other CPU directives.

Support for COMM Directive

386 | ASM now supports the COMM directive for allocating communal variables at link time. A communal definition for a symbol is equivalent to an external definition at assembly time. The only difference is that the linker will allocate the memory for the variable at link time if there has not been a public definition for the symbol. The syntax of the directive is as follows:

```
COMM {NEAR | FAR} name:size{:count}
```

The NEAR or FAR keyword is optional and specifies the segment in which the communal symbol will reside. (See section 6.2.4 of the *386 | LINK Reference Manual* for a complete explanation.) The name parameter specifies the name of the communal symbol. It must follow the same conventions as those for an external symbol. The size parameter specifies the size of each element in the variable. It can be BYTE, WORD, DWORD, PWORD, QWORD, or TBYTE. The count parameter is optional and specifies the number of elements in the variable. If it is omitted, a count of 1 is assumed.

Communal variables are used by C compilers for uninitialized global data which appears in the program's BSS. They do not usually need to be used in assembly language.

.PUBPARSE directive and -PUBPARSE Switch for Public Symbol Table Management

A new directive and command line switch have been added which cause a specified character to be removed from the front of public symbol definition records for the module being assembled. This allows symbols to be referenced by one name inside the module being assembled and by another name from external modules. The name of the command line switch is -PUBPARSE, and its syntax is:

```
-PUBPARSE c
```

The name of the directive is .PUBPARSE, and its syntax is:

```
.PUBPARSE c
```

Both the directive and command line switch take a single argument c which specifies the character to remove from the front of public symbols. This feature is useful to create or reference global symbols which conflict with assembler reserved words. For example, a high level language routine named "mov" could be referenced from assembly language as "_mov" by using the directive ".PUBPARSE _"

Adding Segments to a Previously Defined Group

The GROUP directive can now be used to add additional segments to an existing group. For example, it is now legal to have the following two statements in the same assembly language source file:

```
dgroup  group       dseg
dgroup  group       dseg,stackseg
```

Creating a Text Substitution Symbol

The EQU directive can now be used to create a text substitution symbol with any arbitrary text, including text that resolves to an integer value. To force the EQU directive to create a text substitution symbol, enclose the desired text in angle brackets. For example,

```
textsym equ   <1 + 2 + 3>
```

creates a text substitution symbol with the value "1 + 2 + 3", while

```
constsym equ  1 + 2 + 3
```

creates a constant symbol with the integer value 6.

Support for Sign Extended Immediate Values for Logical Instructions

Two new assembler switches and assembler directives have been added to control the size of the immediate value field for the AND, OR, and XOR logical instructions when they are used with register or memory immediate addressing modes. All Intel CPUs since the 8086 have supported a 1-byte sign-extended form of the immediate operand for full-sized register and memory operands (just like ADD, SUB, etc.), yet the opcodes have never been documented.

The new switches -SIGNLOGI and -NOSIGNLOGI can be used respectively to enable and disable generation of the undocumented form of the instruction. The new directives .SIGNLOGI and .NOSIGNLOGI directives can be used inside a program with the same effect.

If none of the new switches or directives are used, the default operation of 386|ASM is to use the sign-extended form of the immediate value if possible, for compatibility with earlier versions of 386|ASM and Microsoft MASM.

Count Register Size Specification for JCXZ and LOOP Instructions

The JCXZ and JECXZ instructions have been modified to generate address size override bytes when the size of the count register (CX or ECX) differs from the USE attribute (USE16 or USE32) for the code segment. Previously, the assembler incorrectly generated an operand size override byte.

New versions of the LOOP instruction were added which specify the size of the count register to be decremented and tested, regardless of the USE attribute for the segment. When these instructions are used, the assembler will generate override bytes as necessary. The new LOOP instructions which specifically modify the CX register are:

- LOOP16
- LOOPE16
- LOOPZ16
- LOOPNE16
- LOOPNZ16

The new LOOP instructions which specifically modify the ECX register are:

- LOOP32
- LOOPE32
- LOOPZ32
- LOOPNE32
- LOOPNZ32

Incorrect Example Usage of DT Directive

The example of the use of the DUP operator with the DT directive in section 6.3 of the 386 | ASM Reference Manual is incorrect. The DT directive treats all numbers as packed decimal if the current default radix is base 10. Since packed decimal numbers cannot be used as a count for the DUP operator, a radix specifier must be given for the DUP count when using the DT directive. The example in section 6.3 should read:

```
nest1    DT 10d DUP(5d DUP (1.0))
```